Control statements, Loops and Popup boxes in JavaScript

Objectives : At the end of this lesson you shall be able to
- **explain control statements**
- **discuss about various Loops**
- **explain the uses of Popup boxes.**

Control Statements: When we write code for a particular program, we sometimes takes various decisions for executing different action. These can be done through conditional/control statements.

In JavaScript we have the following conditional statements:

Use **if** to specify a block of code to be executed, if a specified condition is true

Use **else** to specify a block of code to be executed, if the same condition is false

Use **else if** to specify a new condition to test, if the first condition is false

Use **switch** to specify many alternative blocks of code to be executed.

The if Statement

Use the if statement to specify a block of JavaScript code to be executed if a condition is true.

Syntax

```
If (condition) {
   block of code to be executed if the condition is true
}
```

Example 1

Make a "Good day" greeting if the time is less than 18:00:

```
if (time < 18) {
   greeting = "Good day";
}
```

The **result** of greeting will be:

Good day

The else Statement

Use the else statement to specify a block of code to be executed if the condition is false.

```
if (condition) {
   block of code to be executed if the condition is true
} else {
   block of code to be executed if the condition is false
}
```

Example 2

If the time is less than 18:00, create a "Good day" greeting, otherwise "Good evening":

```
if (time < 18) {
   greeting = "Good day";
} else {
   greeting = "Good evening";
}
```

The **result** of greeting will be:

Good day

The else if Statement

Use the else if statement to specify a new condition if the first condition is false.

Syntax

```
if (condition1) {
   block of code to be executed if condition1 is true
} else if (condition2) {
   block of code to be executed if the condition1 is false and condition2 is true
} else {
   block of code to be executed if the condition1 is false and condition2 is false
}
```

Example 3

If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 18:00, create a "Good day" greeting, otherwise a "Good evening":

```
if (time < 10) {
   greeting = "Good morning";
} else if (time < 18) {
   greeting = "Good day";
} else {
```

```
  greeting = "Good evening";
}
```

The **result** of x will be:

Good day

The JavaScript Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

Syntax

```
switch(expression) {
  case n1:
    code block
    break;
  case n2:
    code block
    break;
  default:
    default code block
}
```

This is how it works:

- The switch expression is evaluated once.
- The value of the expression is compared with the values of each case.
- If there is a match, the associated block of code is executed.

Example 4

Use today's weekday number to calculate weekday name: (Sunday=0, Monday=1, Tuesday=2, ...)

```
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
    break;
}
```

The **result** of day will be:

Tuesday

The break Keyword

When the JavaScript code interpreter reaches a break keyword, it breaks out of the switch block.

This will stop the execution of more execution of code and/or case testing inside the block.

The default Keyword

The default keyword specifies the code to run if there is no case match:

Example 5

If today is neither Saturday nor Sunday, write a default message:

```
switch (new Date().getDay()) {
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
    break;
  default:
    text = "Looking forward to the Weekend";
}
```

The **result** of text will be:

Looking forward to the Weekend

Common Code and Fall-Through

Sometimes, in a switch block, you will want different cases to use the same code, or fall-through to a common default.

Note from the next example, that cases can share the same code block and that the default case does not have to be the last case in a switch block:

Example 6

```
switch (new Date().getDay()) {
    case 1:
    case 2:
    case 3:
   default:
       text = "Weekend is coming";
       break;
 case 4:
    case 5:
      text = "Weekend is soon";
       break;
    case 0:
    case 6:
      text = "Now in Weekend";
}
```

JavaScript Loops

Loops are handy, if you want to run the same code over and over again, each time with a different value.

Often this is the case when working with arrays:

Instead of writing:

```
text += train[0] + "<br>";
text += train [1] + "<br>";
text += train [2] + "<br>";
text += train [3] + "<br>";
text += train [4] + "<br>";
text += train [5] + "<br>";
```

You can write:

```
for (i = 0; i < train.length; i++) {
    text += train [i] + "<br>";
}
```

Different Kinds of Loops

JavaScript supports different kinds of loops:

- for - loops through a block of code a number of times
- for/in - loops through the properties of an object
- while - loops through a block of code while a specified condition is true
- do/while - also loops through a block of code while a specified condition is true

The For Loop

The for loop is often the tool you will use when you want to create a loop.

The for loop has the following syntax:

```
for (statement 1; statement 2; statement 3) {
    code block to be executed
}
```

Statement 1 is executed before the loop (the code block) starts. It is called Initialisation Part

Statement 2 defines the condition for running the loop (the code block).It is called condition part.

Statement 3 is executed each time after the loop (the code block) has been executed. It is called increment/ decrement part.

Example 7

```
for (i = 0; i < 5; i++) {
    text += "The number is " + i + "<br>";
}
```

From the example above, you can read:

Statement 1 sets a variable before the loop starts (var i = 0).

Statement 2 defines the condition for the loop to run (i must be less than 5).

Statement 3 increases a value (i++) each time the code block in the loop has been executed.

Initialisation Part

Normally you will use statement 1 to initiate the variable used in the loop (var i = 0).

This is not always the case, JavaScript doesn't care. Statement 1 is optional.

You can initiate many values in statement 1 (separated by comma):

Example 8

```
for (i = 0, len = train.length, text = ""; i < len; i++) {
    text += train [i] + "<br>";
}
```

And you can omit statement 1 (like when your values are set before the loop starts):

Example 9

```
var i = 2;
var len = train.length;
var text = "";
for (; i < len; i++) {
    text += train [i] + "<br>";
}
```

Condition Part

Often statement 2 is used to evaluate the condition of the initial variable.

This is not always the case, JavaScript doesn't care. Statement 2 is also optional.

If statement 2 returns true, the loop will start over again, if it returns false, the loop will end.

If you omit statement 2, you must provide a break inside the loop. Otherwise the loop will never end. This will crash your browser. Read about breaks in a later chapter of this tutorial.

Increment/Decrement Part

Often statement 3 increases the initial variable.

This is not always the case, JavaScript doesn't care, and statement 3 is optional.

Statement 3 can do anything like negative increment (i--), or larger increment (i = i + 15), or anything else.

Statement 3 can also be omitted (like when you increment your values inside the loop):

Example 10

```
var i = 0;
len = train.length;
for (; i < len; ) {
    text += train [i] + "<br>";
    i++;
}
```

For/In Loop

The JavaScript for/in statement loops through the properties of an object:

```
var person = {fname:"Raja", lname:"Sen", age:35};
var text = "";
var x;
for (x in person) {
    text += person[x];
}
```

While Loop

The while loop loops through a block of code as long as a specified condition is true.

Syntax

```
while (condition) {
    code block to be executed
}
```

In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

Example 11

```
while (i < 10) {
    text += "The number is " + i;
    i++;
}
```

If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser.

The Do/While Loop

The do/while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax

```
do {
    code block to be executed
}
while (condition);
```

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

Example 12

```
do {
    text += "The number is " + i;
    i++;
}
while (i < 10);
```

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

Comparing For and While

If you have read the previous chapter, about the for loop, you will discover that a while loop is much the same as a for loop, with statement 1 and statement 3 omitted.

The loop in this example uses a for loop to collect the car names from the train array:

Example 13

```
train = ["Duronto","Satabdi","Garib Rath","Rajdhani"];
var i = 0;
var text = "";
for (;train[i];) {
    text += train[i] + "<br>";
    i++;
}
```

The loop in this example uses a while loop to collect the car names from the train array:

```
train = ["Duronto","Satabdi","Garib Rath","Rajdhani"];
var i = 0;
var text = "";
while (train[i]) {
text += train[i] + "<br>";
    i++;
}
```

The Break Statement in Loop

Break statement is used to terminate a loop before its completion. It saves machine time for not iterating a loop uselessly.

For example: In linear search, if we find the item then we can break the loop as no point of runnign it unnecessary.

Example 14

```
for(i=0;i<l;i++ {
        if(arr[i]==item) {
                alert("Found at :"+i);
                fl=1;
                break;
                }
if(fl==0) alert("Not Found");
```

Here, if the item is found, loop breaks and CPU time is saved.

Popup Boxes

JavaScript has three kind of popup boxes. They are

1 Alert box

2 Confirm box and

3 Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user. When an alert box pops up, the user will have to click “OK” to proceed.

Syntax

window.alert(“*sometext*”);

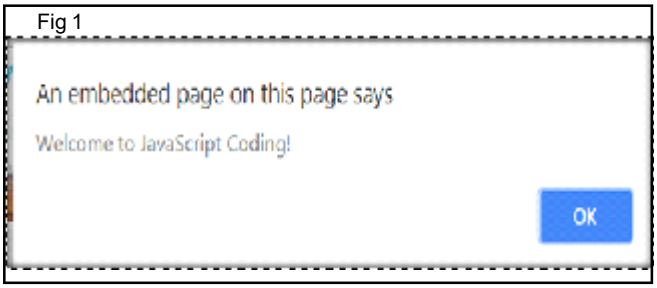
Note: The window.alert() method can be written without the window prefix.

Example 15

alert (“Welcome to Java Script Coding!;)

The result is shown in Fig 1.

Fig 1



Confirm Box

A confirm box is often used to verify or accept something.When a confirm box pops up, the user will have to click either “OK” or “Cancel” to proceed.If the user clicks “OK”, the box returns **true**. If the user clicks “Cancel”, the box returns **false**.

Syntax

window.confirm(“*sometext*”);

Note: The window.confirm() method can be written without the window prefix.

Example 16

```
if (confirm(“Click a button!”))
{
    txt = “ You clicked OK!”;
}
else
{
    txt = “You clicked Cancel!”;
}
```

The result is is shown in Fig 2.

Fig 2

Note: When click on OK button it displays the message "You clicked OK!" and when click on Cancel button it displays the message "You clicked Cancel!"

Prompt Box

A prompt box is often used if the user to input a value before entering a page.When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

window.prompt("*sometext*","*default text*");

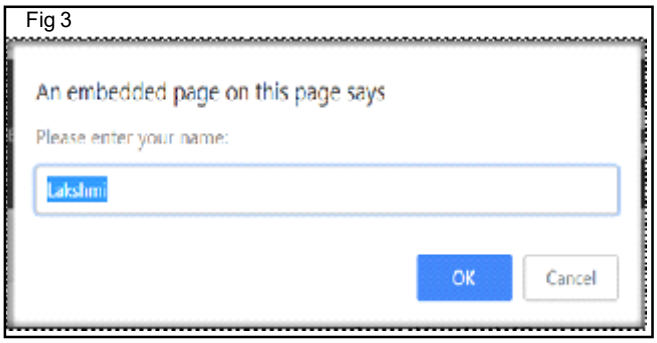
Note: The window.prompt() method can be written without the window prefix.

Example 17

```
var tname = promp t("Please Enter your Name", "Lakshmi");
    if (tname == null || tname == "")
      {txt = "User cancelled the prompt.";
    }
    else
       {txt = "Hello " + tname + "! Congratulations!!!!!");
    }
```

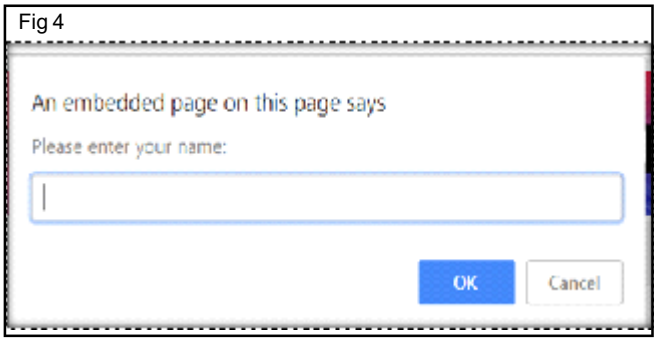
The result is is shown in Fig 3.

Fig 3



Note: If click on OK button it displays the message "Hello Lakshmi! Congratulations!!!!!" If cancelled the name 'Lakshmi' as shown in Fig 4 it gives the message "User cancelled the Prompt". Also whenclick the Cancel button even when if the box has text 'Lakshmi' it gives the message "User cancelled the Prompt".

Fig 4



Line Breaks

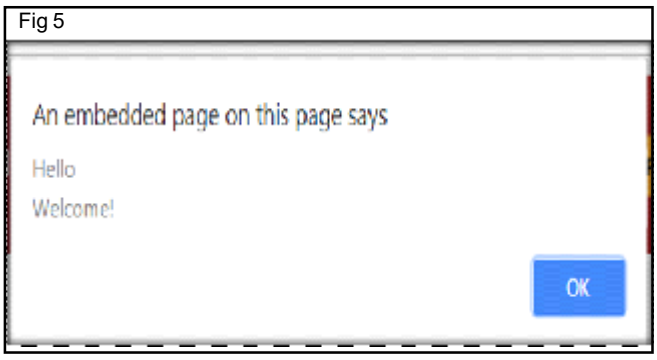
To display line breaks inside a popup box, use a back-slash followed by the character n.

Example 18

alert("Hello\nWelcome!");

The result is shown in Fig 5.

Fig 5



VBA Message boxes and Input boxes

Objectives: At the end of this lesson you shall be able to
- **state the uses of message boxes and input boxes in VBA**
- **describe the msgbox method and msgbox function**
- **describe the inputbox method and inputbox function.**

Introduction

Many applications depend on data input from users to take the necessary action. Excel VBA has very useful functions that allow you to gather user input for your applications. VBA allows you to create message boxes, user input forms and input boxes to get user input.VBA message boxes provide a way to give information to a user and get information from a user while the program is running. The input Box function can be used to prompt the user to enter a value.

Message Box

In VBA Message Boxes fall into two basic categories, the MsgBox method and the MsgBox function.

The MsgBox Method

The message box method is used to display a pre- defined message to the user. It also contains a single command button "OK" to allow the user to dismiss the message and they must do so before they can continue working in the program.

The basic form of the Message Box (msgbox) in VBAis :Msgbox("message")

Example:

```
Sub result()

Msgbox("congratulations")

End sub
```

This displays a message box as shown in Fig 1

Fig 1

Customize the buttons in a VBA message box

The Msgbox() can be customized by changing the buttons and icons placed on it.

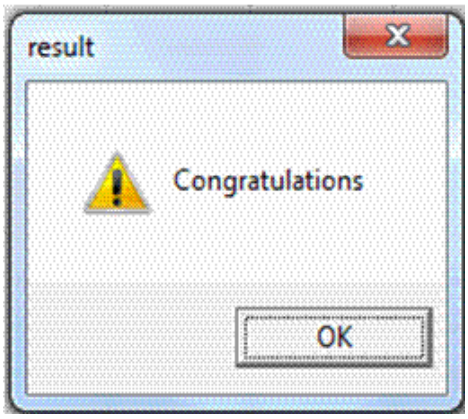
A list of various buttons and icons that can be used in the VBA message box is shown in the Table 1.

For ex. to add an icon and a title to the Msgbox() we can write the following code

```
Sub test()

Dim n As Integer

n = MsgBox("Congratulations", vbExclamation, "result")

End Sub
```

This will produce the following result as in Fig 2.

Fig 2



The MsgBox Function

The MsgBox Function displays a message in a dialog box, waits for the user to click a button, and then returns an integer indicating which button was clicked by the user.The syntax of the Msgbox() function is :

Return value = MsgBox(Prompt, Button and Icon types, Title, Help File, Help File Context)

Table 1

Constant	Description
vbOKOnly	It displays a single OK button
vbOKCancel	It displays two buttons OK and Cancel.
vbAbortRetryIgnore	It displays three buttons Abort, Retry, and Ignore.
vbYesNoCancel	It displays three buttons Yes, No, and Cancel.
vbYesNo	It displays two buttons Yes and No.
vbRetryCancel	It displays two buttons Retry and Cancel.
vbCritical	It displays a Critical Message icon.
vbQuestion	It displays a Query icon.
vbExclamation	It displays a Warning Message icon.
vbInformation	It displays an Information Message icon.
vbDefaultButton1	First button is treated as default.
vbDefaultButton2	Second button is treated as default.
vbDefaultButton3	Third button is treated as default.
vbDefaultButton4	Fourth button is treated as default.
vbApplicationModal	This suspends the current application till the user responds to the message box.
vbSystemModal	This suspends all the applications till the user responds to the message box.
vbMsgBoxHelpButton	This adds a Help button to the message box.
VbMsgBoxSetForeground	Ensures that message box window is foreground.
vbMsgBoxRight	This sets the Text to right aligned
vbMsgBoxRtlReading	This option specifies that text should appear as right-to-left.

Where:

Return Value: Indicates the action the user took when the message box was shown to him/her.

Prompt : It is the message contained in the main body of the message box.

Button and Icon Types : This specifies the set of buttons & Icons and their placement as they would appear to the user.

Help File : This is the path to a help file that the user can refer to on this topic.

Help File Context : This is the pointer to that part of the help file that specifically deals with this message.

Values returned by MsgBox Function:

VBA MsgBox function returns a value based on the user input. These values can be anyone of the ones shown in Table 2.

A Msgbox function example is shown in the code mentioned below.

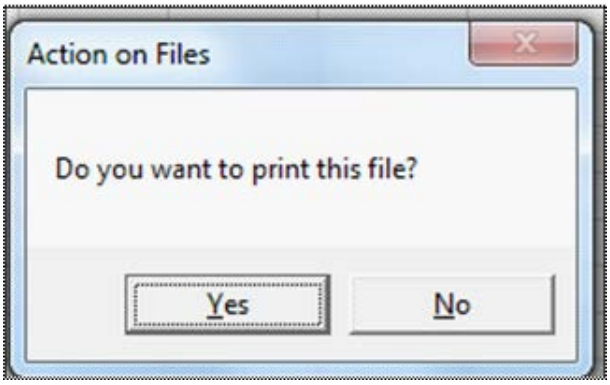
```
Sub test()

Dim n As Integer

 n = MsgBox("Do you want to print this file?", vbYesNo,
"Action on Files")

End Sub
```

Table 2

Value	Description
1	Specifies that OK button is clicked.
2	Specifies that Cancel button is clicked.
3	Specifies that Abort button is clicked.
4	Specifies that Retry button is clicked.
5	Specifies that Ignore button is clicked.
6	Specifies that Yes button is clicked.
7	Specifies that No button is clicked.

This will produce the result as in Fig 3.

Fig 3

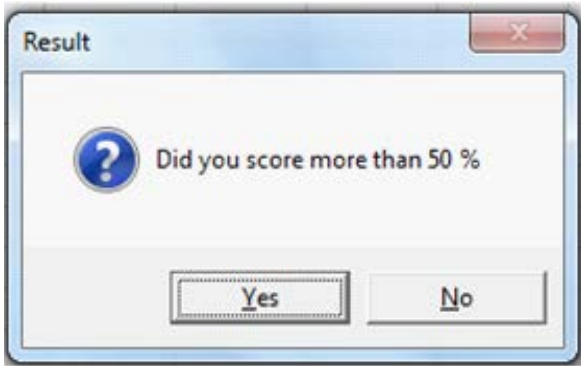


Reading the Msgbox() return values

Based on the value returned by the MsgBox(), decisions can be made.

For ex, the code mentioned here will display the message box, and when the user clicks "Yes" it will display a congratulatory message. If the user clicks "No" another message "Better Luck Next time" will appear as shown in Fig 4.

Fig 4




```
Sub test()

Dim n As Integer

n = MsgBox("Did you score more than 50 % ", vbYesNo +
vbQuestion, "Result")

If n = 6 Then

MsgBox ("Congratulations")

Else

MsgBox ("Better Luck Next Time")

 End If

End Sub
```

Input box

For accepting the input from the user the Input box is used in two ways- The Input Box Function and the Input Box Method.The InputBox method differs from the InputBox function in that it allows selective validation of the user's input, and it can be used with Microsoft Excel objects, error values, and formulas.

Note that Application.Input Box calls the Input Box method; Input Box with no object qualifier calls the InputBox function.

Input Box Function

The Input Box Function displays a dialog box for user input. It returns the information entered in the dialog box. The syntax for the InputBox function is:

InputBox(prompt[, title] [, default] [, xpos] [, ypos] [, helpfile, context])

In its simplest form , the input box function looks like:n = Inputbox("Enter your Age")

The InputBox Method

When we precede the Input Box Function with "Application" we get an InputBox Method that will allow us to specify the type of info that we can collect. Ie. Application.InputBox

Its Syntax is :Input Box(Prompt, Title, Default, Left, Top, HelpFile, HelpContextId, Type)

The Prompt, Title and Default are the same as in the InputBox Function. However, it is the last argument "Type" that allows us to specify the type of data we are going to collect. These are as shown below.

Type:=0 A formula

Type:=1 A number

Type:=2 Text (a string)

Type: = 4 A logical value (True or False)

Type: = 8 A cell reference, as a Range object

Type: = 16 An error value, such as #N/A

Type := 64 An array of values

The following is an example of an InputBox method

```
Sub test()

Dim n As Integer

n = Application.InputBox("Enter you age", "Personal Details", , , , , 1)

 'Exit sub if Cancel button used

If n > 60 Then

MsgBox "You are eligible for senior citizen's concession"

Else

MsgBox ("No concession")

End If

End Sub
```

Looping statements in VBA

Objectives: At the end of this lesson you shall be able to
- **describe the "for" loops in VBA**
- **describe the "do" loops in VBA**
- **explain the use of the "exit" statement in VBA loops**
- **write appropriate code to perform repetitive tasks.**

Introduction

There may be many situations where you need to perform a task repeatedly / a certain number of times. In such cases the code for the task is placed inside a loop and the program iterates or repeats through the loop a certain number of times ie. till a certain condition is met. Some examples of such repetitive tasks are:

a Printing a text or number n number of times.

b Generating a sequence or series of numbers.

c Generating a table of certain calculations.

d Searching / Re arranging a set of numbers etc.

VBA provides the following types of loops to handle looping requirements (Refer Table 1)

Table 1

Loop Type	Description
for next loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
do....until loop	Repeats a statement or group of statements until a condition is met.
do….while loop	Repeats a statement or group of statements as long as the condition is true.

The For Loop

The For ... next loop sets a variable to a specified set of values, and for each value, runs the VBA code inside the loop. For Ex.

```
For n = 1 To 10

debug.print  n

Next n
```

In this example, the initial value of n is set to 1, and the loop code, ie. printing the value of n is performed.The value of n is set to the next value which is by default an increment of 1. Thus this loop is executed 10 times and would print the numbers 1 to 10. The for statement in the above code is the same as For n = 1 To 10 Step 1 since the default increment is 1

The same code will print numbers from 10 to 1 if the step is changed to a negative value as shown below.

```
For n = 10 To 1 Step -1

debug. print  n

Next n
```

Similarly, the following Ex. would add all the numbers from 1 to 10 and print the sum.

```
Dim n, sum as integer

Sum=0

For n = 1 To 10

sum=sum + n

debug. print  sum

Next n
```

The For Each Loop

The For Each loop is similar to the For ... Next loop but, instead of looping through a set of values for a variable, it loops through every object within a set of objects. The following example would print the names of all the worksheets.

```
Dim ws As Worksheet

For each ws in Worksheets

debug. print ws.name

Next ws
```

The Exit For Statement

If you need to end the For loop before the end condition is reached or met, simply use the END FOR in conjunction with the IF statement. In the example given below, we exit the for loop prematurely and before the end condition is

met. The for example given below, the loop exits when n reaches a value of 5.

```
For n = 0 To 10

debug.print n

If n=5 Then Exit For

Next n
```

The Do ….Until Loop repeats a statement or group of statements until a condition is met.

There are 2 ways a Do Until loop can be used in Excel VBA Macro code.

a Test the condition before executing the code in the loop

b Execute the code in the loop and then test for the condition.

Do Until….. Loop

In this example, the value of n is tested before going into the loop.

If the condition n=10 is not met right at the beginning itself, the code inside the loop is not executed at all. The control then jumps to the statements appearing after the Loop statement.

```
Do Until n=10

Debug.print n

n=n+1

Loop
```

Do ….. Loop Until

In this example, the code in the loop is executed at least once before testing the condition. If the condition is true, the looping stops, else the loop is executed again.

```
Do

Debug. print n

n=n+1

Loop Until n=10
```

The Do While … Loop repeats a statement or group of statements as long as the condition is true.

Like the Do until loop, a Do While loop can be also be used in two ways.

a Test the condition before executing the code in the loop

b Execute the code in the loop and then test for the condition.

Do While ….Loop

In this example, the condition ie. num<10 is checked before entering the loop. Only if the condition is met, the code in the loop is executed, otherwise it is skipped entirely. This example will print a table as shown in Fig 1.

Fig 1

number	spc(3)
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100

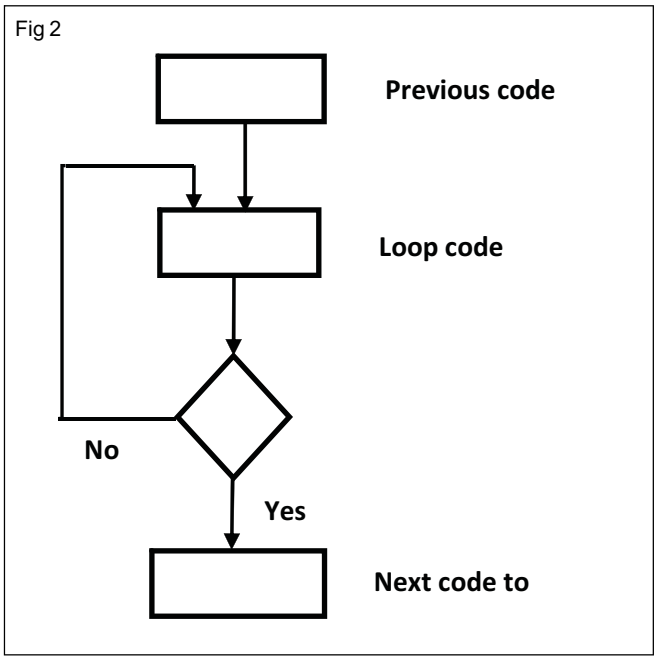
```
Dim num As Integer

Debug.Print "number"; Spc(2); "square"

Do While num < 10

num = num + 1

Debug.Print num; Spc(5); num * num

Loop
```

Do…. Loop While

In a Do…. Loop While , a set of statements in the loop are executed once, then the condition is checked. The code in the loop is executed only if the condition is met. (Refer Fig 2 for the flow chart)

In this example, the value 1 is placed in cell (1,1). The row value is incremented each time the loop code is executed. The incremented value is placed in the cell (row,1). The loop is executed as long as the row value is less than 10 after which the iterations stop. The condition checking is done after executing the loop code at least once. (Fig 2)

Fig 2




```
Dim row As Integer

row = 0

Do

row = row + 1

Cells(row, 1) = row

Loop While row < 10
```

The While ….. Wend loop

The While ….. Wend loop executes a series of statements as long as a given condition is True.

In this example the condition checking is done at the beginning of the loop. This code prints hello 5 times and then prints the value of the counter, ie. 5 at the end of the program.

```
Dim Counter

Counter = 0

While Counter < 5

Counter = Counter + 1

Debug.Print "hello"

Wend

Debug.Print Counter
```

The Exit Statement

The Exit Statement exits a procedure or block and transfers control immediately to the statement following the procedure call or the block definition. It may be in the form of Exit Do, Exit For, Exit While, Exit Select etc. depending on where it is being used. An example of an Exit statement is as follows:

```
Do While True

Count = Count + 1

Debug.Print Count

If Count = 5 Then

Debug.Print "stop at 5"

Exit Do

End If

Loop
```

In this example, the loop condition stops the loop when count=5.